Remix : une architecture pour la recherche dans les masses de données indexées

Gilles Georges, Steven Derrien, Stéphane Rubini, Frédéric Raimbault, Laurent Amsaleg, Dominique Lavenier

ReMIX : une architecture pour la recherche dans les masses de données indexées *

G. Georges[1] – S. Derrien [1] – S. Rubini[1] – F. Raimbault[2] – L. Amsaleg[1] – D. Lavenier[1]

[1] IRISA / INRIA - CNRS, Rennes
[2] Université de Bretagne Sud, Vannes

Résumé

Le projet ReMIX a pour objectif de proposer une architecture matérielle destinée à accélérer la recherche d'information dans les masses de données. Son cœur est une mémoire de très grande capacité étroitement couplée à des ressources reconfigurables. On peut alors accéder rapidement aux données (comparativement à des supports de stockage de type disque magnétique) et envisager des stratégies de recherche innovantes orientées, notamment, sur l'indexation des données. La programmation du système ReMIX est prise en charge par un *framework* qui guide le concepteur depuis la spécification d'une nouvelle application jusqu'à sa mise en œuvre sur le support reconfigurable. Cet article présente les motivations, l'architecture et l'environnement de programmation du système ReMIX.

1. Introduction

La recherche par similarité consiste à retrouver, dans une base de données, un ou plusieurs objets *proches* d'un objet requête.

Ce type de recherche se fonde en général sur une notion de distance entre objets (plus la distance est faible plus la similarité entre objets est forte), le résultat de la recherche étant alors le (ou les) objet(s) possédant la distance la plus faible par rapport à l'objet requête. Le nombre d'objets à retenir peut être soit pré-déterminé (les N objets possédant la distance la plus faible), soit flexible (tous les objets dont la distance est inférieure à un seuil donné).

Bien qu'il permette d'obtenir une qualité de résultats élevée, ce type de recherche se fait en général par un parcours exhaustif de la base d'objets. Ce parcours complet peut s'avérer être très coûteux en temps, soit à cause du nombre d'accès mis en jeu, soit à cause de la charge de calcul liée à l'évaluation des distances entre objets. Incidemment, parce que le temps de recherche évolue linéairement avec la taille de la base, cette approche pose des problèmes pour de grandes masses de données.

Pour éviter un parcours systématique, les objets peuvent être indexés à partir de leurs propriétés. Une recherche consiste alors à extraire les propriétés de la requête, puis à ne solliciter que les objets qui possèdent tout ou partie de ces propriétés.

Pour illustrer, on peut prendre l'exemple (simplifié) de l'indexation d'un texte : on va construire

* Le projet ReMIX est financé par l'ACI Masses de Données (2003 - 2006)

un index qui possède N entrées correspondant aux N mots différents répertoriés dans le texte. Chaque entrée de l'index (ici un mot) pointe vers une liste de phrases (les objets) où ce mot est présent.

Une recherche dans cette base indexée peut par exemple avoir pour objectif de retrouver toutes les phrases similaires à une phrase requête. En d'autres termes retrouver des phrases qui possèdent à peu près les mêmes mots et dans le même ordre. Ainsi, au lieu de passer en revue l'ensemble des phrases du texte, on ne sélectionne que celles qui comportent les mots recherchés. Le nombre de phrases à traiter est donc ainsi considérablement réduit.

Bien que le gain puisse-t-être énorme par rapport à un parcours exhaustif, il faut cependant compter sur les temps d'accès aux données (ou aux objets) qui peuvent être extrêmement pénalisant dès lors que la taille de la base est supérieure à la capacité de la mémoire centrale de la machine, et que cette base doit donc être stockée sur un disque magnétique. En effet, la nature même de l'indexation implique un accès aléatoire aux objets et une requête peut solliciter l'analyse d'un nombre important d'objets. Dans ce cas, le gain par rapport à un accès séquentiel n'est pas garanti.

Afin de bien comprendre l'impact de ce temps d'accès aux données, nous proposons une modélisation (très simplifiée) qui permet de mettre en evidence, dans le cas d'un stockage sur disque magnétique, cette influence sur les performances réelles d'une indexation.

Si on suppose un recouvrement complet entre les temps de traitement et temps d'accès aux objets on peut estimer le temps d'exécution pour un parcours systématique (t_{scan}), ou une indexation (t_{index}) par :

$$t_{scan} = K \times \max \left\{ \begin{array}{l} S/BP \\ t_{dist} \end{array} \right. \tag{1}$$

$$t_{index} = P \times \max \left\{ \begin{array}{l} t_{acces} + n_{obj} \times S/BP \\ n_{obj} \times t_{dist} \end{array} \right. \tag{2}$$

Dans cette formulation, K représente le nombre d'objets de la base, P le nombre moyen d'accès à l'index pour chaque requête, n_{obj} le nombre moyen d'objets associés à chaque entrée de l'index, S la taille moyenne d'un objet, t_{access} le temps d'accès au disque magnétique, t_{dist} le temps pour calculer une distance entre objets, et BP la bande passante entre le dispositif de stockage et les unités de traitement.

Prenons l'exemple d'une base de 10^6 objets (K) de taille égale à 100 octets (S). Supposons que le temps pour calculer la distance entre 2 objets soit égal à 2×10^{-6} seconde (t_{dist}), qu'une requête par indexation nécessite 10^3 accès à l'index (P), soit 1% de la base et qu'une entrée de l'index est associée en moyenne à 5 objets (n_{obj}). Considérons ensuite que la base (comme sa version indexée) est stockée sur un disque possédant une bande passante de 50 Mo/s (BP) et un temps d'accès disque de 10 ms (t_{access}).

Dans ce cas, nous obtenons un temps d'exécution pour l'indexation très supérieur à celui relatif à un parcours systématique de la base : $t_{scan} = 2$ secondes et $t_{index} = 20$ secondes.

Il apparaît donc que, dans le cas de l'utilisation de l'indexation, deux paramètres soient à considérer attentivement :

– le temps d'accès : il devient critique dès lors que la taille de la base dépasse la taille de la mémoire centrale et que celle-ci doit, par conséquent être stockée sur un support de mémorisation de type disque. C'est par exemple le cas lorsqu'on considère des masses de données telles que les banques génomiques, les banques d'images, les banques vidéo, les données multimédia du web, etc.
– la sélectivité : c'est le rapport P/K. Plus ce rapport est faible, plus le nombre d'objets à analyser diminue. L'enjeu d'une indexation efficace est donc de réduire au maximum la valeur de ce rapport. En d'autres termes, les propriétés utilisées pour l'indexation doivent être les plus discriminantes possibles pour permettre de ne sélectionner qu'un petit nombre d'objets, et minimiser à la fois les temps d'accès et les temps liés aux calculs de distance.

L'objectif du projet ReMIX est donc de proposer une architecture matérielle capable :

1. **d'accéder rapidement et aléatoirement à un grand nombre d'objets.** Le but est de rendre possible la mise en œuvre d'algorithmes d'indexation efficaces sur un volume de données conséquent (plusieurs centaines de Go). Ainsi, plutôt que de stocker l'information sur support magnétique, ReMIX propose d'utiliser une mémoire de type Flash de grande capacité (512 Go) qui dispose d'un temps d'accès de l'ordre de 20 μs.
2. **d'exécuter rapidement les calculs de distance.** Ces calculs se prêtent en général très bien à une mise en œuvre matérielle sur de la logique programmable, laquelle est capable d'exploiter efficacement le parallélisme à grain fin disponible dans ces traitements. La mémoire Flash est donc directement connectée à des composants FPGA Xilinx Virtex-II Pro de grande capacité permettant d'implanter des opérateurs de calcul de distance complexes et fonctionnant au rythme du flot de données en provenance de la mémoire.

La mémoire dispose par ailleurs d'un mécanisme de lecture aléatoire anticipée qui permet, si on connaît à l'avance les objets qui vont être accédés, de recouvrir le temps de transfert d'un objet avec le temps d'accès de l'objet suivant. Ainsi, accès, transferts et traitements peuvent opérer en recouvrement. Le temps de calcul t_{index} sur cette architecture s'approxime alors par :

$$t_{index} = P \times max \begin{cases} t_{access} \\ n_{obj} \times S/BP \\ n_{obj} \times t_{dist} \end{cases} \quad (3)$$

En reprenant l'exemple précédent, le temps d'exécution d'une recherche utilisant l'indexation, sur cette architecture, est réduit à $t_{index} = 0.02\,\text{seconde}$.

Cette approche s'inspire des projets RDISK [12], *Smart Disk* [6], *Active Disk* [7, 5] ou *Intelligent Disk* [4] où les ressources de calcul sont connectées au plus près des dispositifs de stockage. La différence majeure réside dans la possibilité d'accéder rapidement et aléatoirement à l'information, ce qui autorise une algorithmique beaucoup plus sophistiquée pour manipuler l'information. Dans tous les cas, l'idée directrice est de filtrer au plus tôt l'information, de manière à ne faire remonter que les données pertinentes vers l'application.

La section suivante présente globalement l'architecture de ReMIX ainsi que les choix technologiques effectués. La section 3 décrit l'environnement de programmation à partir d'un *framework* orienté vers les applications d'indexation. La section 4 indique les premiers résultats obtenus sur cette architecture.

2. L'architecture ReMIX

ReMIX se présente sous la forme d'un cluster de 4 nœuds de calcul commandés par une machine hôte (nœud frontal). Chaque nœud de calcul est un PC enrichi de deux cartes *mémoires reconfigurables* RMEM[2] d'une capacité de 64 Go chacune et disposant de ressources reconfigurables conséquentes pour effectuer des traitements à la volée en sortie de la mémoire. Le système complet dispose donc d'une mémoire de 512 Go associée à quelques 240 000 cellules logiques.

La mémoire étant l'élément central de ce projet, nous détaillons maintenant les motivations des choix technologiques de la mémoire avant d'aborder la description détaillée de la carte RMEM.

2.1. Le choix de la technologie mémoire

L'objectif de ReMIX est de fournir une mémoire de très grande taille disposant d'un temps d'accès aléatoire le plus faible possible par rapport à un disque magnétique. Parmi les technologies mémoire disponibles, nous avons retenu les technologies de RAM statiques, dynamiques, et les technologies de mémoire Flash. Le tableau ci-après résume leurs principales caractéristiques en 2006. Les calculs de nombre de composants, de coût, de bande passante et de consommation sont fait sur la base de la réalisation d'une mémoire 64 bits de 64 Go.

Technologie	Composants nécessaires	Coût mémoire	Temps d'accès	Bande passante	Consommation globale
SRAM	7280	123 500 €	5 ns	800 Mo/s	5250 W
SDRAM	512	4 115 €	10 ns	2 Go/s	30 W
Flash-NAND	64	1 030 €	25 us	160 Mo/s	450 mW
Flash-NOR	4096	72 500 €	100 ns	320 Mo/s	550 mW

La technologie SRAM, si elle offre le temps d'accès le plus court, souffre d'un coût par unité de stockage extrêmement élevé. Ainsi, la mise en œuvre d'une mémoire de 64 Go à base de SRAM nécessiterait 7 280 composants pour un coût de revient prohibitif de l'ordre de 123 500 € et une consommation électrique supérieure à 5 KW.

Les mémoires DRAM offrent en revanche un coût par unité de stockage beaucoup plus abordable. Cette amélioration de la densité de stockage se fait cependant aux dépens des temps d'accès qui sont généralement plus importants (de l'ordre de la dizaine de nanosecondes).

Si cette technologie semble être, à priori, un choix naturel pour implanter une mémoire de très grande taille, une telle mise en œuvre pose de nombreux problèmes. Tout d'abord, ces mémoires, qui se présentent en général sous la forme de modules DIMM (intégrant une capacité de stockage de l'ordre de 1 Go), sont difficiles à intégrer en grand nombre sur un circuit imprimé : le nombre très élevé de broches d'entrées/sorties rend le routage d'autant plus difficile que ces composants fonctionnent à des fréquences très élevées (jusqu'à 400 MHz pour les composants les plus rapides).

La dernière alternative repose sur l'utilisation de mémoires en technologie Flash. Celles-ci se divisent en deux familles : les Flash-NAND et les Flash-NOR. Si la technologie Flash-NOR offre un temps d'accès aléatoire faible (de l'ordre de 100 ns) et un usage proche de celui des mémoires SDRAM, sa densité d'intégration reste limitée (128 Mb). Comme pour la technologie SRAM, l'intégration d'une mémoire de très grande taille est difficilement envisageable.

[2] RMEM : Reconfigurable MEMory

Les Flash-NAND présentent, quant à elles, des densités d'intégration supérieures à celles des RAM dynamiques (on trouve en 2006 des composants Flash-NAND de 32 Gb contre 1 Gb pour la SDRAM). Si l'on s'intéresse au coût par unité de stockage, il apparaît que la technologie Flash-NAND est de très loin la plus avantageuse. Ce constat est confirmé par les tendances de ces dernières années qui suggèrent que l'écart de coût entre technologie DRAM et Flash va continuer à s'accroître en faveur de cette dernière.

La principale différence entre les Flash-NAND et les autres technologies mémoires porte sur le mode d'accès aux données. Dans le cas des Flash-NAND, celui-ci se fait selon un mode paginé avec une latence de chargement de page de 20 μs, soit près de 1000 fois plus que pour une technologie SDRAM. Cette latence d'accès reste malgré tout très inférieure à celle d'un disque magnétique, puisque l'on a cette fois-ci un facteur 1000 en faveur de la mémoire Flash.

Il se pose malgré tout la question de l'impact de cette latence d'accès sur les performances d'une mémoire d'index réalisée à base de composants Flash-NAND. Comme il l'a été montré dans l'introduction, le temps d'accès aléatoire t_{access} joue un rôle prédominant dans la durée de traitement d'une requête dans une base indexée.

Son impact peut cependant être réduit si la quantité de données transférée à chaque accès est suffisamment importante pour atténuer l'effet de la latence d'accès. C'est le cas dans la très grande majorité des applications : d'une part la taille d'un objet n'est en général pas négligeable (entre 10 et 100 octets), d'autre part un accès à l'index signifie souvent un accès à plusieurs objets (paramètre n_{obj}).

Il est par ailleurs possible, dans certaines situations, de recouvrir la phase de chargement d'une page avec la phase de transfert des données d'une autre page : L'architecture interne des composants Flash-NAND autorise le fonctionnement concurrent de plusieurs composants connectés à un même bus. Ce recouvrement n'est cependant possible que lorsque les pages accédées successivement sont allouées à des composants mémoire distincts.

Reprenons l'exemple donné dans l'introduction, en considérant deux technologies mémoires, disposant d'une même bande passante $BP = 500$Mo/s mais caractérisées par des temps d'accès différents $t_{access1}$=20 ns, et $t_{access2}$=20 μs. En se basant sur l'équation 3, on obtient, t_{index_1}=0.2 seconde pour la mémoire à accès rapide et t_{index_1}=0.1 seconde pour la mémoire à accès lent.

On constate ainsi, que malgré un temps d'accès 1000 fois supérieur à celui d'une mémoire de type SDRAM, les performances obtenues à partir d'une mémoire de type Flash sont très largement supérieures à celle obtenues pour une recherche sur disque dur, et restent assez proche de celles obtenues à partir d'une mémoire SDRAM. Il nous semble donc que l'utilisation de mémoire Flash-NAND s'avère donc être un choix pertinent dans le cadre de ReMIX. Ce choix impose cependant deux contraintes :

- **La gestion des blocs défectueux** : les composants Flash-NAND sont organisés en blocs qui forment l'unité minimale d'accès à la mémoire en écriture et en effacement. La technologie Flash ne garantit pas la validité de tous les blocs dans un composant. A titre d'exemple, une mémoire de 512 Mo s'organise en 4096 blocs, avec un maximum de 80 blocs défectueux. La présence de ces blocs défectueux oblige à utiliser un espace d'adressage logique en sus de l'adressage physique, afin de permettre d'utiliser cette mémoire comme un espace d'adressage linéaire de blocs valides.
- **La durée de vie limitée de la mémoire** : les opérations d'effacement de blocs altèrent le support de stockage de l'information. De ce fait, les mémoires Flash ne supportent qu'un

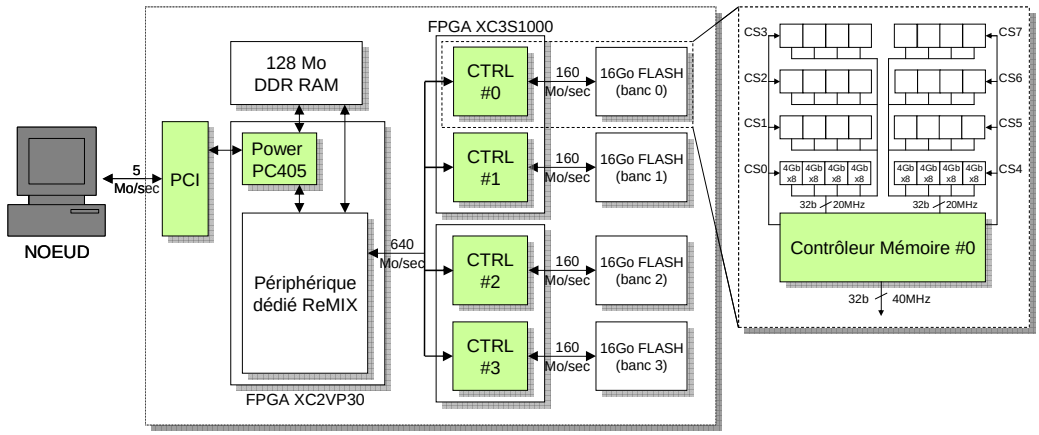

FIG. 1 – Architecture d'une carte RMEM

nombre limité de cycles d'effacement par bloc (environ 10^5 cycles). Passée cette limite, l'écriture correcte des données n'est plus garantie dans le bloc. L'affectation des données dans la mémoire lors des écritures et/ou lors des modifications de blocs doit donc être soigneusement considérée afin de maximiser la durée de vie de la mémoire. Une technique de *Wear-leveling* a été mise en place dans ce but (voir la section 2.3). Toutefois, le nombre limité de cycles d'effacement a un impact négligeable dans le cadre d'utilisation de la mémoire ReMIX, c'est à dire en média de stockage d'information : en supposant une mise à jour quotidienne de l'ensemble de la mémoire ReMIX, les 10^5 cycles d'effacement ne seront atteints qu'au bout de 273 années !

2.2. Architecture de la carte RMEM

La carte RMEM est le cœur du système ReMIX. Sa conception a eu pour objectif d'associer sur un même support une mémoire de grande taille et des ressources reconfigurables utilisées pour réaliser un traitement à la volée des données au fur et à mesure de leur lecture. La figure 1 décrit son architecture au niveau système. La carte RMEM se base sur une carte de développement pour FPGA, à laquelle nous avons associé une carte d'extension mémoire Flash-NAND.

La carte de développement est basée sur une interface PCI et dispose d'un FPGA Virtex-II Pro, associé à 128 Mo de SDRAM. Le Virtex-II Pro embarque l'équivalent de 30 000 cellules logiques, 136 blocs mémoire de 18 Kb et deux cœurs de processeurs PowerPC405.

La carte mémoire est composée 64 boîtiers mémoire de 1 Go, chaque boîtier intègre deux composants Flash qui peuvent être utilisés de manière concurrente. Ceux-ci sont regroupés en 4 bancs indépendants de 16 Go, chaque banc étant organisé en une matrice de 8×4 composants dans laquelle on peut accéder à 8 composants en parallèle. Chaque carte mémoire contient également deux circuits FPGA utilisés pour implémenter les contrôleurs des 128 composants Flash, et pour gérer l'interface entre chacun des 4 bancs mémoire et le circuit Virtex-II Pro de la carte PCI.

Le Virtex-II Pro intègre une architecture de type système sur puce construite autour du processeur PowerPC et d'un bus CoreConnect. Le système intègre plusieurs périphériques (contrôleur mémoire SDRAM, DMA, etc.) connectés au bus système, parmi lequel un contrôleur dédié aux échanges de données (lecture et écriture) entre la carte mémoire Flash et la mémoire SDRAM de la carte PCI.

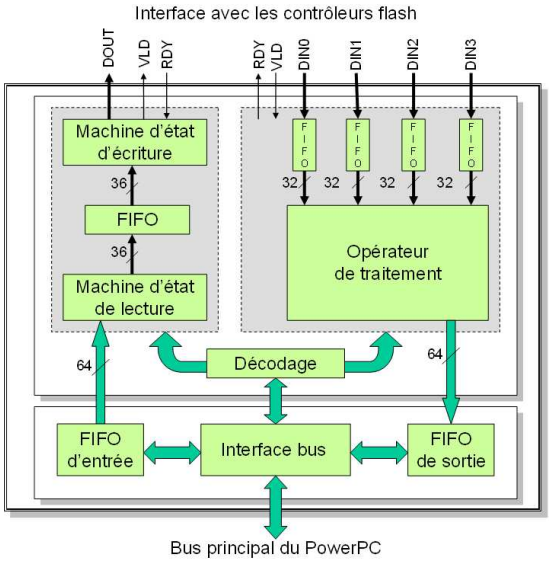

FIG. 2 – Schéma de principe du périphérique dédié ReMIX

Le rôle de ce périphérique d'interface, dont l'architecture est représentée en figure 2, et qui sera désigné sous le terme de périphérique ReMIX dans la suite de cet article, est double :
- Il assure d'une part la gestion des opérations de lecture et d'écriture des données de/sur la mémoire Flash.
- Il intègre d'autre part un bloc *opérateur* matériel spécialisé pour chaque application, dont le rôle est de traiter à la volée les données lues à partir de la mémoire Flash.

L'opérateur reconfigurable peut recevoir simultanément 4 flux de données (4×32 bits) en provenance des 4 bancs de la mémoire Flash et produit en résultat un flux de données sur 64 bits. La synchronisation, en entrée comme en sortie, est assurée par des mémoires de type FIFO. 15 000 cellules logiques et 90 blocs mémoire RAM de 18 Kb sont disponibles pour implémenter l'opérateur spécialisé ReMIX, dont le fonctionnement est directement contrôlé par le PowerPC au travers du bus système.

Grâce à l'accès en parallèle à plusieurs composants Flash, la bande passante mémoire en entrée de l'opérateur est de 640 Mo/s, soit deux ordres de grandeur de plus que celle disponible sur l'interface PCI de la carte ReMIX. Le bus PCI est donc un goulot d'étranglement, qui limite l'intérêt d'une utilisation directe de cette mémoire par l'hôte, puisque l'on dispose alors d'une bande passante très réduite (inférieure par un ordre de grandeur à celle d'un disque dur), ce qui réduit sensiblement les gains liés au temps d'accès aléatoire faible de la Flash (toujours par rapport à un disque).

En pratique, ce type d'échange est exclu : le principe de fonctionnement de ReMIX consiste à utiliser l'opérateur matériel pour filtrer les données directement à la source et ainsi ne transmettre que l'information utile. Dans ce contexte, et en supposant que le taux de filtrage de l'opérateur est suffisant, la bande passante délivrée par l'interface PCI est suffisante. On peut résumer le déroulement d'un traitement sur cette architecture, c'est-à dire la recherche et le traitement d'un objet dans la base d'index, comme suit :

1. Un ou plusieurs ordres de lecture sont envoyés depuis l'hôte (via le processeur PowerPC)

aux contrôleurs Flash par le périphérique d'interface ReMIX.

2. L'opérateur ReMIX est paramétré par l'hôte (toujours via le PowerPC) en vue du traitement des données qui seront lues à partir de la mémoire Flash.
3. Les données lues, disponibles dans les FIFOs à l'entrée de l'opérateur, sont alors filtrées à la volée par ce dernier, qui génère un flux de résultats.
4. Ce flux résultat est transféré vers la mémoire SDRAM par l'utilisation d'un DMA, pour être finalement envoyé à l'hôte par l'interface PCI.
5. Une fois le traitement terminé, l'opérateur signale la fin des calculs au PowerPC par un signal d'interruption.

2.3. Intégration système de la carte RMEM

Afin d'offrir une abstraction du fonctionnement de la carte RMEM aux couches logicielles supérieures, le PowerPC intègre un système d'exploitation réduit qui offre des services d'accès :

- à la mémoire Flash par des primitives d'accès logiques (lecture d'une page dans un bloc logique, écriture/modification/effacement d'un bloc logique).
- à l'opérateur reconfigurable ReMIX par des primitives permettant de paramétrer et de superviser le fonctionnement de l'opérateur.

Il assure de fait la gestion des blocs défectueux des composants Flash en maintenant en mémoire une table de translation d'adresses permettant de faire le lien entre un numéro de bloc logique (associé à une adresse logique dans la mémoire ReMIX) et un numéro de bloc physique sur un (ou plusieurs) composants Flash. Par ailleurs, le système met en œuvre une technique de *Wear-leveling* de premier niveau dans le but de minimiser le nombre d'opérations d'effacement lors de l'écriture ou la modification de blocs logiques. Cette technique repose sur une allocation circulaire des blocs physiques libres Flash [13].

Les cartes RMEM sont intégrées au système sous la forme d'un périphérique PCI. Le pilote PCI interagit directement avec la couche système mise en œuvre sur le PowerPC, et fournit une interface d'accès au système de gestion de fichiers (voir paragraphe suivant) pour les accès mémoire, et au *framework* (voir la section suivante) pour le contrôle et la reconfiguration de l'opérateur.

Un système de fichiers spécifique à ReMIX a par ailleurs été développé. Celui-ci gère automatiquement l'allocation des données sur les différents bancs mémoire Flash. Le but est d'optimiser leur taux d'occupation et ainsi maximiser l'usage de la mémoire globale. Chaque carte RMEM dispose de son propre système de fichiers.

Cet ensemble de fonctionnalités système est exploité par un environnement de programmation (*framework*) de haut niveau qui permet de gérer de manière simplifiée la distribution des traitements, et la répartition des bases d'index sur les différents nœuds ReMIX. La section suivante présente cet environnement de programmation.

3. L'environnement de programmation

L'environnement de programmation ReMIX est proposé dans le but de simplifier le développement d'algorithmes de recherche par similarité dans des masses de données indexées. Il a été conçu avec deux impératifs majeurs : simplicité et réutilisabilité.

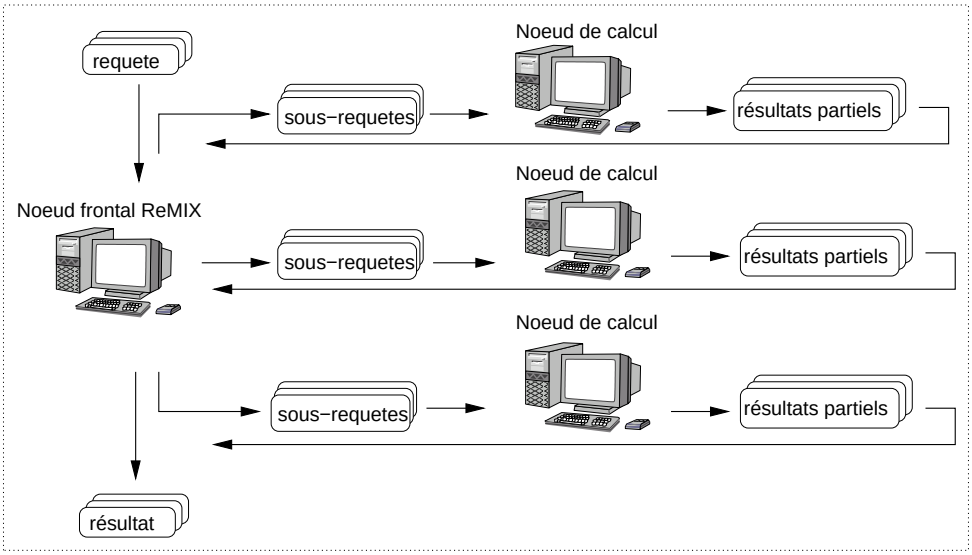

FIG. 3 – Principe du modèle client/serveur du système ReMIX

La simplicité de conception découle de l'utilisation d'un modèle d'exécution de type client/serveur de requêtes distribué. Dans ce modèle, la distribution des données est explicite et statique ; l'exécution, les communications et les synchronisations sont implicitement gérées. L'ensemble s'inscrit dans un *framework* (cadre de conception).

La réutilisabilité de l'environnement de programmation s'inscrit à deux niveaux : logiciel et matériel. D'une part, l'environnement n'est pas lié à un domaine d'applications et, d'autre part, l'environnement ne dépend pas d'une plateforme d'exécution particulière ; sa configuration est précisée explicitement dans un fichier. Il faut spécifier le nombre de nœuds, la nature de la plateforme (cluster de PC, multiprocesseur ou station de travail unique), le support de mémorisation (mémoire RAM, Flash ou disque dur) et le support d'exécution des opérateurs de calcul (FPGA, circuits dédiés ou thread). Outre sa généricité, cette approche permet une mise au point progressive des applications. On peut débuter par des tests sur une station de travail unique, puis passer progressivement à l'échelle sur un système parallèle (grappe de PC) et terminer par une mise en œuvre matérielle des opérateurs sur le système ReMIX. Chaque étape est un raffinement de la précédente.

Il faut cependant noter que la conception automatique des opérateurs sur la structure reconfigurable de ReMIX n'est pas prise en charge : une interface système assure simplement les échanges entre le processeur d'un nœud de calcul et l'opérateur de la carte RMEM. L'opérateur doit être conçu indépendamment.

Le reste de cette section décrit succinctement l'implantation actuelle qui s'appuie sur un mode de fonctionnement de type client/serveur de requêtes et sur l'environnement Java.

3.1. Le modèle client/serveur de requêtes

La figure 3 schématise le modèle de fonctionnement. On suppose que les objets sont répartis sur chaque nœud de calcul et que seul le nœud frontal à connaissance de la répartition. En d'autres termes, il possède l'index permettant de retrouver l'ensemble des objets.

Toute requête soumise au nœud frontal est découpée en sous-requêtes. Chaque sous-requête est ensuite aiguillée vers un des nœuds de calcul. Ces derniers traitent de manière indépendante leurs sous-requêtes. En règle générale, l'opération consiste en un calcul de distance. Chaque nœud de calcul renvoie ses résultats vers le nœud frontal qui les fusionne et délivre en retour le résultat final. Le fonctionnement de l'ensemble est asynchrone et, en régime de croisière, de multiples requêtes peuvent être traitées simultanément.

3.2. Le framework

Le choix d'un mode de fonctionnement unique ayant été retenu, un *framework* contenant toutes les fonctionnalités communes à toutes les applications s'est imposé. L'ensemble a été réalisé selon une approche orientée objet dans l'environnement de programmation standard Java. Le *framework* contient donc deux types de classes : celles qui relèvent du fonctionnement général du modèle d'exécution mis en place et celles qui sont spécifiques à l'application. Parmi ces classes, le programmeur doit, par exemple, spécifier :

- la description des entrées de l'index ;
- la répartition des objets ;
- la manière de découper une requête en sous requêtes ;
- le traitement d'une sous-requête ;
- le post traitement à l'issu de la réception du résultat des sous requêtes ;

Le traitement d'une sous requête est relatif à l'opérateur qui doit être implanté dans le FPGA. Sa description sous la forme d'une méthode Java permet de tester l'application dans un environnement standard avant d'en concevoir une version matérielle. Aucun changement n'est apporté entre les deux implantations : dans le cas de l'utilisation d'une carte RMEM, le *bitstream* est chargé au préalable dans le FPGA et la méthode Java est supplanté à l'exécution par des appels au système d'interface de la carte RMEM.

3.3. Intégration des applications

Trois applications sont actuellement étudiées dans le cadre du projet ReMIX : une application de recherche dans une banque d'images, une version indexée du logiciel phare en génomique, le programme BLAST [1] [3], et une application de recherche d'information dans des banques de données XML. Ces développements laissent présager plusieurs profils d'utilisateurs :

1. pour un domaine d'application connu : porter un nouvel algorithme consiste à programmer le cluster de PC en Java à l'aide du *framework*, à charger et à exploiter des opérateurs matériels prédéfinis sur le FPGA. Ce type d'utilisation est accessible au programmeur spécialiste de son domaine applicatif ;
2. pour un nouveau domaine d'application : en plus des développements pré-cités, il s'agit de concevoir de nouveaux opérateurs de calcul, dans un premier temps en Java, puis de les synthétiser à l'aide des outils de CAO classiques. Ce type d'utilisation est réservé aux architectes de machine ;
3. pour une nouvelle architecture : le *framework* a été conçu pour être portable et d'autres cibles sont envisagées. Par exemple, il est possible, grace au mécanisme d'héritage des langages à objet, de remplacer le support d'exécution et de mémorisation des index dans des unités mobiles dans un contexte d'informatique ambiante ou encore d'utiliser des disques spécialisés pour passer à l'échelle du peta-octet.

4. Conclusion

Le système ReMIX est opérationnel depuis le printemps 2006. Il est actuellement composé d'un cluster de 5 machines : l'une d'elle joue le rôle de maître et coordonne l'exécution de 4 autres machines qui renferment chacune 2 cartes RMEM. L'ensemble représente un espace de stockage cumulé de 512 Go de mémoire Flash auquel est directement connecté une puissance de calcul conséquente sous la forme de ressources reconfigurables. Par rapport à des travaux similaires, notre approche se démarque par la possibilité d'accéder rapidement aux données, comparativement à un disque magnétique, ce qui permet d'envisager de nouvelles techniques de recherche basées sur l'indexation.

En parallèle, un environnement de programmation a été développé. Il s'appuie sur un *framework* spécialement étudié pour l'indexation de grosses masses de données. Il peut s'exécuter sur différentes plateformes parallèles permettant ainsi le développement progressif d'applications, du prototypage jusqu'à l'intégration effective d'opérateurs reconfigurables spécialisés. Le système est actuellement en phase de test, à la fois sur les aspects matériel et logiciel. Dans ce cadre, nous travaillons plus précisément sur deux domaines applicatifs : l'image et la génomique.

L'environnement de programmation a été entièrement validé sur la recherche d'images par le contenu [11]. Une requête est une image et on veut récupérer toutes les images *similaires* (ou parties d'images similaires) parmi des millions d'images d'une base de données. Les images de la base subissent d'abord un traitement pour exhiber certaines propriétés qui, d'un point de vue purement informatique, s'expriment par une collection de vecteurs attachés à chaque image. Un calcul de distance entre images revient alors à calculer des distances entre vecteurs. Cette application a été complètement intégrée dans le *framework* et (automatiquement) parallélisée sur le cluster ReMIX. La dernière étape consiste à synthétiser l'opérateur de distance qui, pour l'instant, est exécuté par logiciel.

La validation matérielle des cartes RMEM s'appuie, quant à elle, sur une application génomique en partenariat avec l'équipe INSERM U694 du CHU d'Angers. Cette équipe cherche à identifier de nouvelles protéines mitochondriales qui peuvent être impliquées dans un grand nombre de processus pathologiques tels que : cancer, diabète, obésité, maladies génétiques et neurodégénératives. On en connaît environ 500 sur les 1500 estimées. Repérer de nouvelles instances est donc primordial pour améliorer la compréhension de la physiopathologie mitochondriale. D'un point de vue bioinformatique, il s'agit de retrouver dans le génome humain des gènes ancestraux communs avec l'ensemble des bactéries séquencées à ce jour. Des similitudes entre 400 000 protéines bactériennes et les 3 milliards de nucléotides qui constituent le génome humain sont à rechercher systématiquement. La mémoire de ReMIX contient l'indexation du génome et le traitement est découpé en deux phases qui correspondent à deux filtrages différents. Pour chaque phase, un opérateur reconfigurable spécifique été développé. L'ensemble est en cours de validation et une estimation raisonnable indique un temps de calcul maximal de 3 jours. Ceci est à comparer aux quelques mois sur un cluster de PC de taille équivalente avec le logiciel BLAST [1] [3].

Ces deux applications constituent une base de travail pertinente. Elles répondent à des préoccupations réelles et permettent de tester le prototype en conséquence. La première expérimentation illustre bien l'approche descendante que nous souhaitons mettre en place dans l'environnement ReMIX : une application est d'abord mise au point à l'aide d'un cadre logiciel spécifique. Le programmeur est guidé dans sa démarche pour aboutir à une spécification précise de ce qui peut être supporté par les cartes RMEM. Dans le *framework*, l'opérateur reconfigurable est explicitement spécifié dans une procédure JAVA.

L'étape suivante est de traduire cette spécification en une architecture matérielle. En fait, dans ReMIX les traitements à réaliser sur les cartes RMEM peuvent être considérés comme un filtrage sur un flot de données continu. C'est donc un cadre relativement bien défini sur lequel on peut s'appuyer pour guider une synthèse architecturale et qui s'insere bien dans une gamme d'outils déjà disponibles [9] [10] [8]. La piste que nous poursuivons consiste à adapter l'outil de synthèse GAUT [2] dans l'environnement ReMIX. GAUT est développé au LESTER, à Lorient, et accepte un code C comme spécification ; il génère une description VHDL que nous pouvons ensuite intégrer dans le périphérique de contrôle dédié ReMIX.

En conclusion, le projet ReMIX est encore en pleine phase de validation. La technologie Flash couplée très étroitement à des ressources reconfigurables laissent présager d'excellentes performances. L'expérience montre cependant qu'un des points bloquants reste souvent la programmation des architectures reconfigurables. Nous essayons, dans ce projet, de développer conjointement un environnement qui permette d'augmenter le cercle des programmeurs potentiels.

Bibliographie

1. S.F. Altschul, W. Gish W, W. Miller, E.W. Myers, D.J. Lipman, Basic local alignment search tool, J Mol Biol. 215(3) :403-10, 1990.
2. E. Martin, O. Sentieys, H. Dubois, J.L. Philippe, GAUT : an Architecture Synthesis Tool for Dedicated Signal Processors, In proceedings of EURO-DAC 93, pp. 14-19, 1993.
3. S.F. Altschul, L.M. Thomas, A.A. Schäffer, J. Zhang, Z. Zhang, W. Miller, D.J. Lipman, Gapped BLAST and PSI-BLAST : a new generation of protein database search programs, Nucleic Acids Res. 25 :3389-3402, 1997.
4. K. Keeton, D. A. Patterson, J. M. Hellerstein, A Case for Intelligent Disks (IDISKs), SIGMOD Record, Vol. 27, No. 3, 1998.
5. A. Acharya, M. Uysal, J. Saltz, Active Disks : Programming Model, Algorithms and Evaluation, ASPLOS-VIII, San Jose, California, 1998.
6. G. Memik, M.T. Kandemir, A. Choudhary, Design and Evaluation of Smart Disk Architecture for DSS Commercial Workloads, In Proceedings of International Conference on Parallel Processing (ICPP), Toronto, Canada, 2000.
7. M. Uysal and A. Acharya and J. Saltz, Evaluation of Active Disks for Decision Support Databases, HPCA-2000, Toulouse, France, 2000.
8. T.J. Callahan, J. Hauser, J. Wawrzynek, The Garp Architecture and C Compiler, IEEE Computer, April 2000.
9. J. Frigo, M. Gokhale, D. Lavenier, Evaluation of the Streams-C C-to-FPGA Compiler : An Application Perspective, 9th ACM International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 2001.
10. B. Draper, W. Bohm, J. Hammes, W. Najjar, R. Beveridge, C. Ross, M. Chawathe, M. Desai, Compiling SA-C Programs to FPGAs : Performance Results, J. Bins.International Conference on Vision Systems, Vancouver, July 7-8, 2001.
11. L. Amsaleg, P. Gros, S-A. Berrani, Robust Object Recognition in Images and the Related Database Problems, Special issue of the Journal of Multimedia Tools and Applications, 23 :221-235, 2004.
12. S. Guyetant, M. Giraud, L. L'Hours, S. Derrien, S. Rubini, D. Lavenier, F. Raimbault, Cluster of re-configurable nodes for scanning large genomic banks, Parallel Computing, 31(1), 2005.
13. E. Gal, S. Toledo : Algorithms and data structures for flash memories. ACM Comput. Surv. 37(2) :138-163, 2005.